Lecture 26: Variable Selection & High-Dimensional Regression

1 What Is Variable Selection?

"Variable selection" means selecting which variables to include in our model. This is useful if there are many covariates. For example, if there are more covariates than data points, i.e., a **high-dimensional regime**, then we know that $\mathbf{X}^T\mathbf{X}$ is singular so the least squares estimator is not well-defined.

This is an increasingly common situation in data analysis. A very large genetic study might sequence the genes of, say, 500 people — but measure 500,000 genetic markers in each person. If we want to predict some characteristic of the people from the genes (say their height, or blood pressure, or how quickly they would reject a transplanted organ), there is simply no way to estimate a model by ordinary least squares. Any approach to high-dimensional regression *must* involve either reducing the number of dimensions or penalizing the estimates such as ridge regression or the lasso.

More generally, there is a **bias-variance tradeoff**. Let $\widehat{m}$ denote our estimated model and let $m(x) = \mathbb{E}[Y|X = x]$ be the true regression function. Let $\overline{m}(x) = \mathbb{E}[\widehat{m}(x)]$. If $\widehat{m}$ was unbiased then $\overline{m}(x) = m(x)$. But, in general, we cannot assume this is true. First of all, the true model might not be linear. But even if it was linear, we could still have bias because, when we select variables, we might be omitting important variables.

In the following calculation, we treat X as random. The prediction error of a future observation (X, Y) is

$$\begin{aligned} R \equiv \mathbb{E}(Y - \widehat{m}(X))^2 &= \mathbb{E}(Y - m(X) + m(X) - \overline{m}(X) + \overline{m}(X) - \widehat{m}(X))^2 \\ &= \tau^2 + B^2 + V \end{aligned}$$

where

$$\begin{aligned} \tau^2 &= \mathbf{E}(Y - m(X))^2 \\ B^2 &= \mathbf{E}[(\overline{m}(X) - m(X))^2] \\ V &= \mathbf{E}[(\widehat{m}(X) - \overline{m}(X))^2]. \end{aligned}$$

In this formula, we call τ^2 the unavoidable error. (Predicting a random variable involves error even if we knew the true function $m(x)$.) B^2 is the (squared) bias term and V is the variance.

Generally speaking: small models (with few covariates) have low variance and high bias. Large models (with many covariates) have high variance and low bias. The challenge in variable selection is to choose a model with small prediction error and this requires that we balance the bias and variance.

Our main tools for high-dimensional regression will be: the lasso, ridge regression and something called forward stepwise regression.

2 Why Variable Selection Using p-Values Is a Bad Idea

When we assume the linear, constant-variance, independent-Gaussian-noise model is completely correct, it is easy to test the hypothesis that any particular coefficient is zero. The (Wald) test statistic is

$$t = \frac{\widehat{\beta}_i}{\widehat{\mathrm{se}}\left[\widehat{\beta}_i\right]}$$

and, under the null hypothesis that $\beta_i = 0$, this has a $t_{n-(p+1)}$ distribution, therefore tending to a z (standard-Gaussian) distribution as $n \rightarrow \infty$.

It is very, very tempting, and common, to use the p-values which come from this test to select variables: significant variables get included, insignificant ones do not, ones with smaller p-values (hence larger test statistics) are higher priorities to include than ones with smaller test statistics. This pattern of reasoning shows up over and over again among users of regression, including, I am ashamed to say, not a few statisticians.

The reasons why this is a bad idea were already gone over in lecture 15, so, again, I will be brief. Let us think about what will tend to make the test statistic larger or smaller, by being more explicit about the denominator:

$$\frac{\widehat{\beta}_i}{\frac{\widehat{\sigma}}{\sqrt{n\widehat{\mathrm{Var}}[X_i]}}\sqrt{VIF_i}}$$

where $\widehat{\mathrm{Var}}\left[X_i\right]$ is the sample variance of the i^{th} predictor variable, and VIF_i is that variables variance-inflation factor (see Lecture 17). What follows from this?

1. Larger coefficients will, all else being equal, have larger test statistics and be more significant ($\widehat{\beta}_i$ in the numerator).

2. Reducing the noise around the regression line will increase all the test statistics, and make every variable more significant ($\widehat{\sigma}$ in the denominator).

3. Increasing the sample size will increase all the test statistics, and make every variable more significant ($\sqrt{n}$ in the denominator).

4. More variance in a predictor variable will, all else being equal, increase the test statistic and make the variable more significant ($\widehat{\mathrm{Var}}\left[X_i\right]$ in the denominator).

5. More correlation between X_i and the other predictors will, all else being equal, decrease the test statistic and make the variable less significant (VIF_i in the denominator).

The test statistic, and thus the p-value, runs together an estimate of the actual size of the coefficient with how well we can measure that particular coefficient. This is exactly the right thing to do if our question is "Can we reliably detect that this coefficient isn't exactly zero?" That is a very, very different question from "Is this variable truly relevant to the response?", or even from "Does including this variable help us predict the response?" Utterly trivial variables can show up as having highly significant coefficients, if the predictor has lots of variance and isn't very correlated with the other predictors. Very important (large-coefficient) variables can be insignificant, when their coefficients can't be measured precisely with our data. *Every* variable whose coefficient isn't exactly zero will eventually (as $n \rightarrow \infty$) have an arbitrarily large test statistic, and an arbitrarily small p-value.

None of this is even much help in answering the question "Which variables help us predict the response?" F-tests on groups of coefficients don't help either. t-tests on individual coefficients.

3 Cross-Validation

The solution is to use cross-validation as we discussed in Lecture 21. You could also use AIC or C_p which are really just approximations to cross-validation. Remember that we had two versions of cross-validation: K-fold, and leave-one-out.

Cross-validation provides us with an estimate of the prediction error

$$R = \mathbb{E}(Y - \widehat{m}(X))^2.$$

So our goal is consider any models and choose the one with the lowest estimated prediction error.

4 How Do We Fit All the Models?

Suppose there are p covariates. For each variable, we can decide to keep it in the model or throw it away. This means that there are 2^p possible models. In principle we could fit all 2^p such models, estimate the prediction error of each one and choose the best. This presents a problem. There are too many models to consider. For example, if $p = 100$ then 2^p is equal to 1,267,650,600,228,229,401,496,703,205,376. That's a lot of models.

There are two common solutions. The first is to use *forward stepwise regression* (also known as greedy regression) and the second is to use the *lasso.*

5 Standardization

Before we proceed, you should know that when doing variable selection it is common practice to standardize the variables. This means that we take each covariate, subtract off its mean and divide by its standard deviation. This makes sure we are comparing variables on a simialr scale. Usually, we also replace Y_i with $Y_i - \overline{Y}$. One consequence of this is that we no longer need an intercept term in the model. The standardize a matrix, you can use the `scale` command in R. However, most of the R programs for variable selection will automatically standardize the variables.

6 Forward Stepwise Regression

Forward stepwise regression works like this:

1. Start by fitting the simplest model $Y = \beta_0 + \epsilon$. Let $S = \emptyset$.

2. Next we consider all single variable models:

$$Y = \beta_0 + \beta_1 X_1 + \epsilon, \quad Y = \beta_0 + \beta_2 X_2 + \epsilon, \quad \cdots .$$

   We fit each of these and then choose the best one. "Best" means, lowest RSS or lowest Cross-validation error, or lowest AIC etc. Add the best variable to S. For example, suppose X_{17} was the best. Then $S = \{17\}$ and our current estimated model is $\widehat{m}(x) = \widehat{\beta}_0 + \widehat{\beta}_{17} X_{17}$.

3. Now consider adding another variable. So in this case we consider

$$Y = \beta_0 + \beta_1 X_1 + \beta_{17} X_{17} + \epsilon, \quad Y = \beta_0 + \beta_2 X_2 + \beta_{17} X_{17} + \epsilon, \quad \cdots .$$

Choose the best one and add it to S. For example, if X_5 was the best then $S = \{5, 17\}$ and our current model estumate is

$$\widehat{m}(x) = \widehat{\beta}_0 + \widehat{\beta}_5 X_5 + \widehat{\beta}_{17} X_{17}.$$

Note the the coefficients have changes. The estimate $\widehat{\beta}_{17}$ you gte here will be differenr than the estimate in the previous step, because we are now fitting a two-variable model.

4. Continue adding one variable at a time this way until you cannot add any more variables.

Each step of the process gives us a new model. We we have a set of models $M_1, M_2, \ldots,$. Now we estimate the prediction error of each model and choose the best one.

6.1 Stepwise Regression R

There are several programs in R for stepwise regression.

The first is to use the `step` command. To use this, you have to start by fitting the smallest and largest model.

```
small = lm(y ~ 1,data=D)  ### fit intercept only
big   = lm(y ~ .,data=D)  ### fit all the variables
tmp = step(small,scope = list(lower=small,upper=big),direction="forward")
```

You should look carefully at `help(step)`. Here is a very small example, with only three variables so you can see what the outout looks like:

```
D = data.frame(y=y,x1=x1,x2=x2,x3=x3)
small = lm(y ~ 1,data=D)
big   = lm(y ~ .,data=D)
tmp = step(small,scope = list(lower=small,upper=big),direction="forward")

Start:  AIC=-1.23
y ~ 1

       Df Sum of Sq    RSS    AIC
+ x1    1    2284.1 3925.9 371.02
+ x2    1    1775.5 4434.5 383.20
+ x3    1    1215.9 4994.1 395.08
<none>              6210.0 414.87

Step:  AIC=371.02
y ~ x1

> small = lm(y ~ 1,data=D)
big   = lm(y ~ .,data=D)
tmp = step(small,scope = list(lower=small,upper=big),direction="forward")
```

```
> Start:  AIC=414.87
y ~ 1

       Df Sum of Sq    RSS    AIC
+ x1    1    2284.1 3925.9 371.02
+ x2    1    1775.5 4434.5 383.20
+ x3    1    1215.9 4994.1 395.08
<none>              6210.0 414.87

Step:  AIC=371.02
y ~ x1

       Df Sum of Sq    RSS    AIC
+ x2    1    1971.6 1954.3 303.26
+ x3    1    1328.2 2597.8 331.72
<none>              3925.9 371.02

Step:  AIC=303.26
y ~ x1 + x2

       Df Sum of Sq     RSS     AIC
+ x3    1    1953.5    0.83 -470.67
<none>              1954.32  303.26

Step:  AIC=-470.67
y ~ x1 + x2 + x3
```

The `step` command uses AIC. Is it the negative of the AIC we used so small values are good. The search may stop early if there is no improvement in AIC.

I prefer to use the lars package. To use lars, you need to create a matrix with all your covariates.

```
library(lars)
out = lars(x,y,type="stepwise")    ### x is my design matrix. y is my outcome.
tmp = cv.lars(x,y,K=10,type="stepwise",plot.it=FALSE)  ##runs stepwise
m = length(tmp$cv)
plot(1:m,tmp$cv)  ### plot the cross-validation scores
## now add error bars to the plot
for(i in 1:m){
      segments(i,tmp$cv[i]-tmp$cv.error[i],i,tmp$cv[i]+tmp$cv.error[i])
      }
j = which.min(tmp$cv)   ### which model minimizes cross-validation
print(j) ###
[1] 2

### print the beta.hat vector for the best model
round(out$beta[j,],2)
```

```
  [1] 3.27 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [16] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [31] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [46] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [61] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [76] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [91] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## we see the best model included x1 but no other variables

## this command tells you at what step each variable was chosen
out$entry
  [1]  1  0  0  0 39 20  0 44 47  0 32 25  6  0 28 29  0 48  0 22  0 26  0 45  4
 [26]  0  0  0  5  0  0  0  3 30 31 35 10 11  0  0  8 38 17  0 36 42 43  0 46 33
 [51] 34 13 37 16  0  0  0  9  0  7  0 23  0  0  0  0 40  0 12 27  0 41  0  0  0
 [76]  0  0  0  0  0  0  2 49  0 19  0  0 14  0 15  0 21  0  0  0  0  0 18 24  0
```

See the plot in Figure 1.

7 Ridge Regression

Earlier in the semester we discussed how *ridge regression* can help with collinearity; it can also help us with high-dimensional regression. Instead of minimizing

$$\frac{1}{n}(\mathbf{Y}-\mathbf{X}\mathbf{b})^T(\mathbf{Y}-\mathbf{X}\mathbf{b})$$

we instead minimize the penalized squared error

$$\frac{1}{n}(\mathbf{Y}-\mathbf{X}\mathbf{b})^T(\mathbf{Y}-\mathbf{X}\mathbf{b})+\frac{\lambda}{n}\|\mathbf{b}\|^2.$$

The **penalty factor** $\lambda > 0$ will lead to a solution with some bias but it reduces the variance. In particular, it solves the problem of non-invertibility. We'll come back later to how to pick λ. The gradient is

$$\nabla_{\mathbf{b}}\left(\frac{1}{n}(\mathbf{Y}-\mathbf{X}\mathbf{b})^T(\mathbf{Y}-\mathbf{X}\mathbf{b})+\frac{\lambda}{n}\mathbf{b}^T\mathbf{b}\right)=\frac{2}{n}\left(-\mathbf{X}^T\mathbf{Y}+\mathbf{X}^T\mathbf{X}\mathbf{b}+\lambda\mathbf{b}\right).$$

Set this to zero at the optimum, $\widehat{\beta}_\lambda$,

$$\mathbf{X}^T\mathbf{Y}=(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})\widehat{\beta}_\lambda$$

and solve to get

$$\widehat{\beta}_\lambda=(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}.$$

The inverse always exists.

Let's compute the mean and variance:

$$\mathbb{E}\left[\widehat{\beta}_\lambda\right] = \left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbb{E}\left[\mathbf{Y}\right] \quad (1)$$

$$= \left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{X}\beta \quad (2)$$

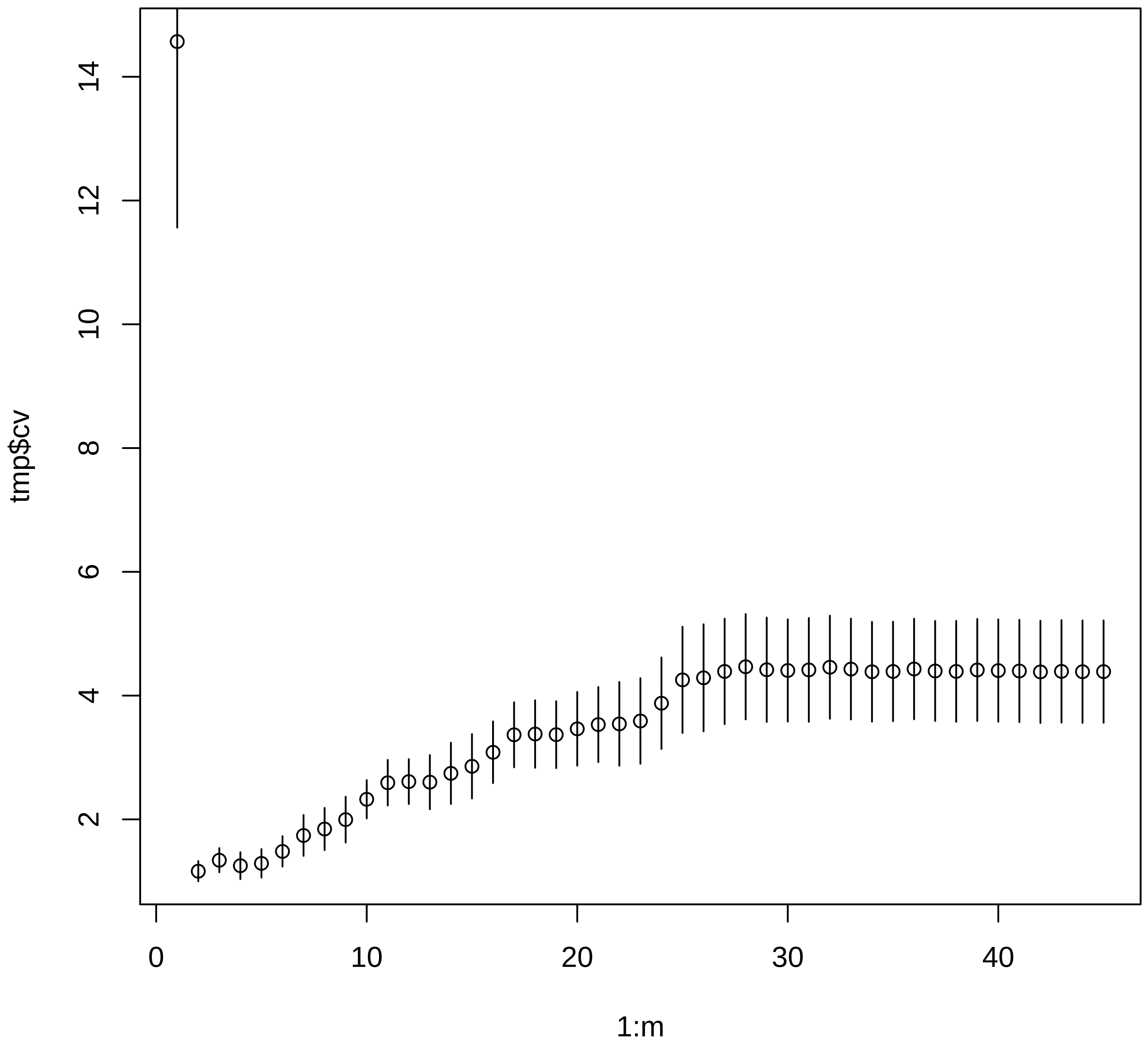

Figure 1: *The cross validation scores as forward stepwise adds variables.*

$$\begin{aligned}
\operatorname{Var}\left[\widehat{\beta}_\lambda\right] &= \operatorname{Var}\left[\left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{Y}\right] && (3)\\
&= \operatorname{Var}\left[\left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\epsilon\right] && (4)\\
&= \left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\sigma^2\mathbf{I}\mathbf{X}\left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1} && (5)\\
&= \sigma^2\left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}. && (6)
\end{aligned}$$

Notice how both of these expressions smoothly approach the corresponding formulas ones for ordinary least squares as $\lambda \to 0$. Indeed, under the Gaussian noise assumption, $\widehat{\beta}_\lambda$ actually has a Gaussian distribution with the given expectation and variance.

It can be shown that ridge regression can also be obtained by doing a constrained minimization:

$$\text{minimize} \quad (\mathbf{Y}-\mathbf{X}\mathbf{b})^T(\mathbf{Y}-\mathbf{X}\mathbf{b}) \quad \text{subject to } ||\mathbf{b}||^2 \leq c.$$

You prove this using Lagrangian multipliers that you learned in calculus.

We usually choose λ using *cross-validation* which we will explain later in the course.

7.1 Ridge Regression in R

There are several R implementations of ridge regression; the `MASS` package contains one, `lm.ridge`, which needs you to specify λ. The `ridge` package has `linearRidge`, which gives you the option to set λ, or to select it automatically via cross-validation.

7.2 Other Penalties/Constraints

Ridge regression penalizes the mean squared error with $\|b\|^2$, the squared length of the coefficient vector. This suggests the idea of using some other measure of how big the vector is, some other **norm**. A mathematically popular family of norms are the ℓ_p norms defined as

$$\|b\|_p = \left(\sum_{i=1}^{p} |b_i|^p\right)^{1/p}.$$

The usual Euclidean length is ℓ_2, while ℓ_1 is

$$\|b\|_1 = \sum_{i=1}^{p} |b_i|$$

and (by continuity $\|b\|_0$ is just the number of non-zero entries in b. When $p \neq 2$, penalizing the $\|b\|_q$ does not, usually, have a nice closed-form solution like ridge regression does. Finding the minimum of the mean squared error under an ℓ_1 penalty is called **lasso regression** or the **lasso estimator**, or just **the lasso**. This has the nice property that it givea *sparse* solutions — it sets coefficients to be exactly zero (unlike ridge). There are no closed forms for the lasso, but there are efficient numerical algorithms. The lasso is one of the most popular methods for high-dimensional regression today. We will discuss the lasso in detail later.

Penalizing ℓ_0, the number of non-zero coefficients, *sounds* like a good idea, but there are, provably, no algorithms for quickly trying all possible combinations of variables. (The problem is NP hard.)

8 The Lasso

A common (and more modern) approach is to use the lasso. We define $\widehat{\beta}$ as the vector that minimizes

$$(\mathbf{Y} - \mathbf{X}\beta)^T(\mathbf{Y} - \mathbf{X}\beta) + \lambda||\beta||_1$$

where $||\beta||_1 = \sum_j |\beta_j|$. This is like ridge regression except the penalty function is the L_1 norm instead of the L_2 norm. It turns out that the estimator $\widehat{\beta}$ is sparse: many of the elements are 0. This corresponds to eliminating these variables from the model. We can use K-fold cross-validaton to choose λ.

8.1 Example

I suggest using the `glmnet` package.

```
library(glmnet)

out = glmnet(x,y)   ### lasso fit
plot(out)           ### plots the betas as lambda varies

cvfit = cv.glmnet(x,y)   ### computes the cross-validation scores
plot(cvfit)              ### nice plot of cv
cvfit$lambda.min         ### best vaue
[1] 0.2455386
coef(cvfit,s="lambda.min")  ### print the coefficients of the best model


(Intercept) -0.06889786
V1           .
V2           2.74342130
V3           .
V4           .
V5           2.93688741
V6           .
V7           .
....   there is more but I am not including here to save space
```

9 Inference after Selection, Again

If all you care about is prediction, you may not need to hypothesis tests or confidence intervals. But if you want to use tests and confidence intervals, then there is a problem. The standard inferential statistics (like the p-values on individual coefficients) are not valid if you do variable selection. The easy cure is to split the data in half at random, and use one part to do model selection and the other half to do inference for your selected model.

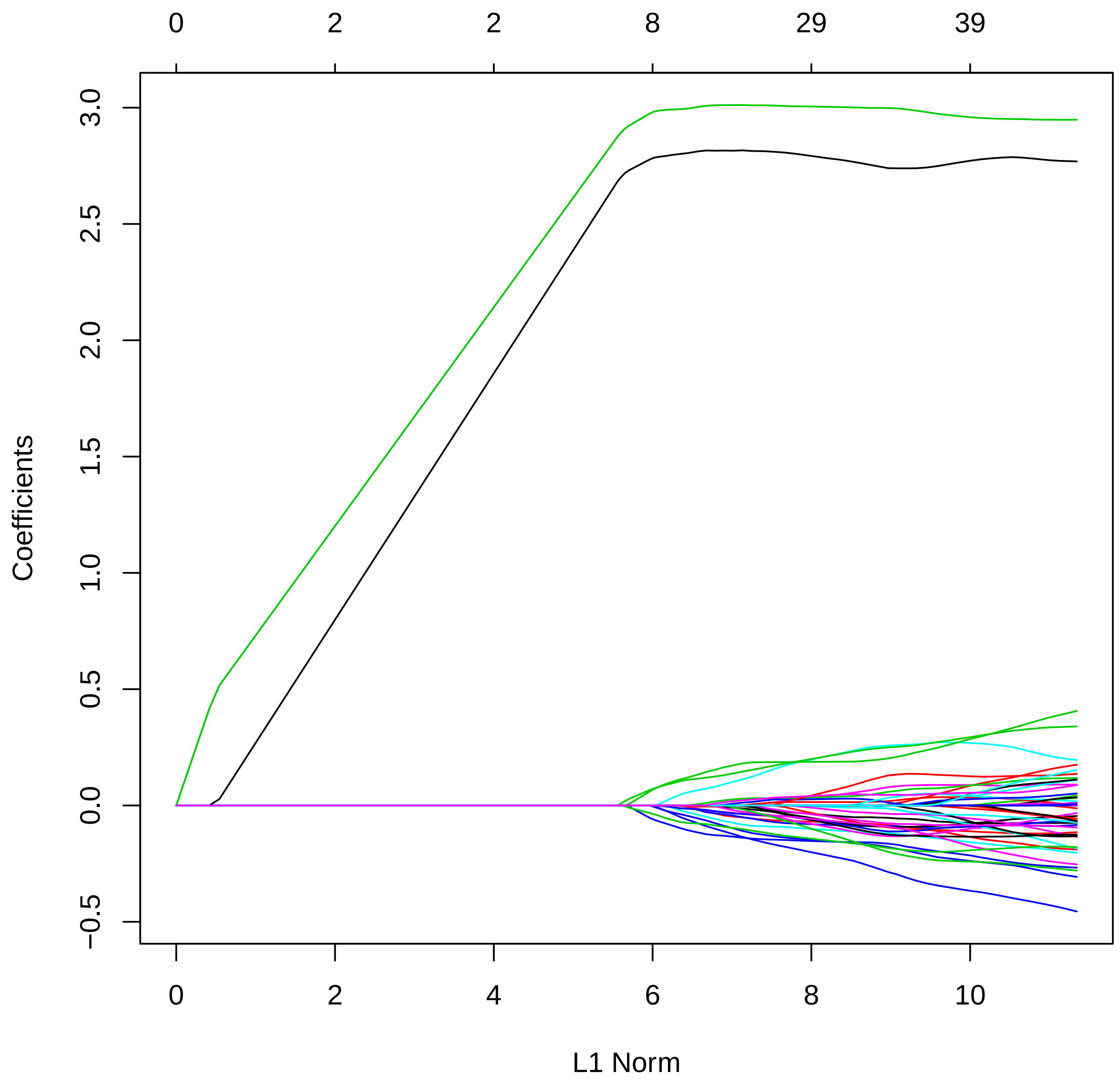

FIGURE 2: *The lasso path. Each curve is one $\widehat{\beta}_j$ as λ varies.*

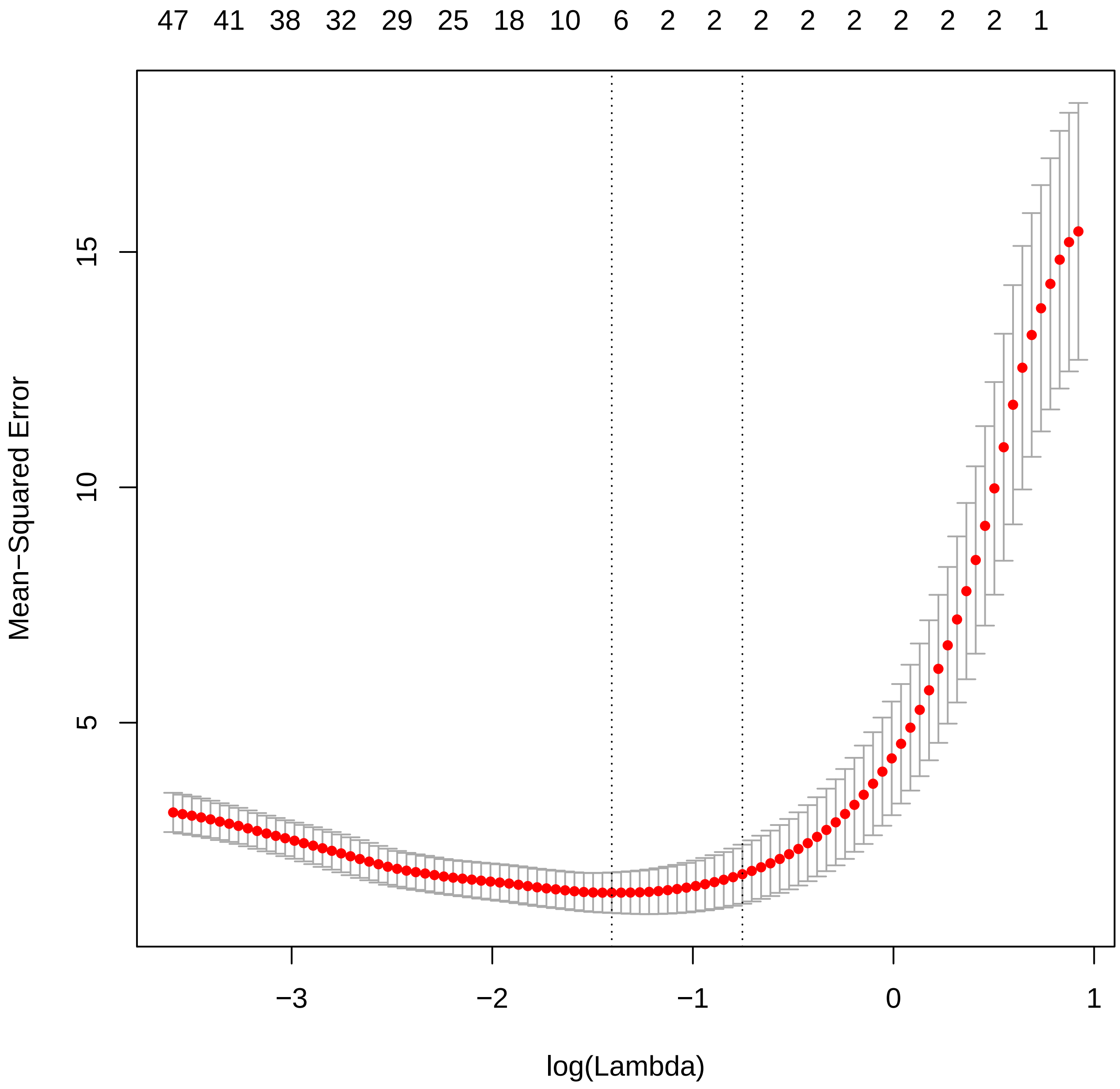

FIGURE 3: *The cross validation scores as λ varies.*